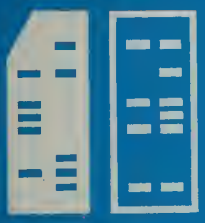
UIUCDCS-R-73-563

THE DESIGN, IMPLEMENTATION AND ANALYSIS OF A COMPUTER-ASSISTED INSTRUCTION SYSTEM ON A MINI-COMPUTER

by

Alan Mark Davis

January, 1973



DEPARTMENT OF COMPUTER SCIENCE
UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN · URBANA, ILLINOIS

UIUCDCS-R-73-563

THE DESIGN, IMPLEMENTATION AND ANALYSIS OF A COMPUTER-ASSISTED INSTRUCTION SYSTEM ON A MINI-COMPUTER

by

Alan Mark Davis

January, 1973

Department of Computer Science
University of Illinois at Urbana-Champaign
Urbana, Illinois

This work was supported in part by the National Science Foundation under Grant No. US NSF GJ-812 and was submitted in partial fulfillment for the Master of Science degree in Computer Science, 1973.

ACKNOWLEDGMENT

The author would like to thank Professor Donald B. Gillies for his advice and support throughout the development of this project and the writing of this thesis. Special thanks also goes to Don Oxley, without whose constant effort to develop ETS and enumerable suggestions for GIZMO, this project would never have come to pass. Bill Holt has also been of enormous help in the implementation of student mode of GIZMO. My thanks also go to Jim Hart, Keith Phillips and Stan Kerr for their development of system software for ETS which assisted and in fact enabled GIZMO to be born. The initial ideas of my CS 201 quiz section in Spring, 1971 and the further development of these ideas by my later quiz sections were a major contribution to my work. Ian Stocks, with his uncanny ability to instantly find bugs in code and Jim Miller, Harold Lopeman, Bob Miller, and Gordy Peterson, with their tireless efforts to keep the hardware working must be congratulated. My thanks also goes to the Engineering Open House participants who were the first set of guinea pigs to use GIZMO. Educational Technology publications has given me permission to reproduce the diagram on page 44.

I am also very grateful to Mrs. June Wingler for her excellent job of typing this paper and her patience throughout the writing of it. Without her help and dedication during the last four weeks before format check, I would have had to wait another six months before completing this. To all of the above people plus Bob and Karen Lantz and Kirsten Trimble I give my thanks for their friendship and moral support throughout this project. Last, but not least, to my parents, for their love and continuous faith in me, and for a few other things without which I know I could not have even started on this project, thanks.

TABLE OF CONTENTS

LIST OF FIGURES

I. INTRODUCTION

A. CAI Systems

The major purpose for computer-assisted instruction (CAI) is to simulate the dialogue that occurs between a student and his teacher. As it becomes more and more widely introduced in many disciplines certain difficulties are arising. In an attempt to create a single CAI system which can complement the human teacher in any field at any level of education, this attempt at generality has also resulted in expensive complicated systems which are fairly difficult to learn to use effectively. It is my thesis to limit the class of problems to be used on a particular system and thus make that system both cheap and easy to learn and easy to use for the teacher as well as the student and systems programmer.

Possible classes of problems for which a CAI system could be aimed toward might include personal (i.e., computer-student) instruction, simulation of laboratory experiments, problem solving, student inquiry, drill and practice, or dynamic supervision of computer program writing [Stolurow, 1968; Salisbury, 1971].

Personal instruction involves the teacher or lesson writer in programming the CAI system to: 1) teach or guide a student through certain lesson material, 2) ask questions, 3) evaluate results, and 4) either review the same or go on with new material depending on these results. These tasks require among other things, an erasable output device such as a Plasma-panel or a CRT display. The particular hardware configuration that I was working

on included teletypes as the only interactive input/output device. It is impractical therefore to handle this class of problems because it is too tempting for the average student to reread the text material which has just been printed on the paper output upon being faced with a question which he cannot answer without difficulty.

Simulation of laboratory experiments once again requires more elaborate hardware than was available. For example, waiting for a teletype to print out a picture of, say, a male fruit fly with vestigial wings could be quite frustrating for the student. Introduction of display panels will increase the burden on the lesson writing language and therefore make it more difficult for a teacher to use the CAI system. By including this class of problems in the CAI system in question, we would also be adding unduly to its cost.

Problem solving is a useful tool considering the speed of a computer's computation. This is the mode where a student must program the computer to perform desired tasks and then feed the program the required data and accept the results. I do not wish to include this class of problems in my discussion as almost all computers can be easily set up for this and does not require any specific educational software interface.

Inquiry mode is where the students ask the system questions and are answered from a large set of answers and algorithms present within the computer's memory. The advantage of this is that the students need not know anything about the CAI system; the communication is through natural language. The major problem with this class of problems is the necessity for the teacher to learn the workings of the CAI system in order to write the usually elaborate routines used to search and locate the required information to answer the student's question.

Drill and practice has been accepted as the main use for GIZMO, the CAI system which is the subject of most of this paper. Drill and practice is easily applicable to a teletype's relatively slow input and output. In addition, it requires only a small set of instructions in its lesson writing language. For example it could well get by with just question statements, correct answers, incorrect answers, and some way of associating a specific computer response to a particular student answer. For GIZMO, a slightly larger superset of this has been chosen. It will be explained in a later chapter.

The problem of a CAI system supervising the construction of a program or program segment, written in either a higher level language or in assembly language introduces some interesting questions: 1) Will it add significantly to the lesson-writing language? 2) Will the system remain inexpensive? The answer to the first question is probably yes. The GIZMO lesson-writing language, which I claim is small and easy to learn, is, just as it stands, ideally suited to write a syntax recognizer. Therefore, it would be easier to find errors at that level. What cannot be done with this easy language is simulate execution of the student's program without giving the teacher the ability to link GIZMO with other previously written modules that could interpret the code. This linking as well as the necessity for the teacher to specify new types of evaluation criteria would add to the size of the language. For these reasons this class of problems, namely, dynamic supervision of program writing was left out of the present version. It is hoped that this added feature will not add significantly to the ultimate cost of the system. It is also hoped that the author will be able to continue researching this particular area.

B. GIZMO

Hicks and Hunka, in their book, The Teacher and the Computer, state an assumption, "Students should aid in the development of CAI. They should not be used solely as guinea pigs[Hicks, 1972]." This philosophy was taken to the extreme when GIZMO was first started. The justification for calling it extreme is that they didn't just aid in this development; they were the developers. In the Spring, 1971 semester, the author was a University of Illinois teaching assistant for CS 201, a course in introductory assembly language and systems programming. For a final project in the course, the students were to design, code, and debug a large-scale programming project under the author's supervision. They chose to design a computer-aided instruction system called GIZMO. In order that future CS 201 classes be able to modify and expand the system, it was necessary to insist on simple structure and strong modularity. To summarize the basic goals established at this initial stage: 1) to fulfill the requirements of the present course by participating in a large project as a programming exercise, 2) to maintain specific internal standards and documentation to enable future CS 201 classes to understand, modify, or expand the system's program to their liking, 3) to provide a CAI system that could be used effectively to assist in the actual teaching of the basic assembly language principles of CS 201, and 4) to provide a CAI system which is extremely easy to use by both the lesson-writer and the student [Davis, 1972].

The present GIZMO bears little resemblance to the original product (except, of course, its name). However, the original goals have been maintained throughout the system's evolution.

GIZMO is now a set of three programs that operate under the operating system called Educational Timesharing System (ETS)[Oxley, 1972]. Here is a brief description of the three programs. The teacher (or author) mode is the program which "leads the teacher by the hand" through the simple steps of writing a lesson in a Backus-Naur-like language. The lesson compiler is the program which translates the output of the teacher mode into a form that is easily interpreted by the student mode. The student mode is the program which communicates a lesson to the student, evaluates his answer, and responds accordingly. Thus, it attempts to mimic the dialog of a student and teacher during a drill and practice session.

GIZMO is designed to be a CAI system which is easy for a teacher to learn to use effectively. Since it was designed predominantly for use in teaching programming language fundamentals, a lesson-writing language employing spelling-correction, key word finders and other heuristics was not as important as one that enabled the teacher to specify many similar but well-defined, precise answers easily. The choice of a BNF-like language, to be explained later, has enabled simplicity, elasticity, and precision to exist at once.

In addition to the above mentioned design criteria, others also came into play at various stages in the development. Here are a few:

The choice of having a compiler was decided upon because we wanted 1) an easily readable and easily editable copy of the lesson (one very similar to actual teacher inputs) and 2) an easily interpretable copy of the lesson to maximize efficiency and response time during student use.

We didn't want to be prejudiced by previous successes or blinded by previous failures of CAI's. We (the CS 201 class and I) thus chose to decide on what we wanted the system to do and how we wanted it to do it. Therefore, we deliberately avoided reading about other in-progress CAI systems until completion of the initial version.

In conclusion, let me say that this thesis is not intended to be a treatise on lesson-writing, on educational philosophy, or on what a "perfect" CAI system is. It is simply a description and analysis of one particular CAI system with the above mentioned design criteria, all directed toward the ultimate goal of simplicity and cost-effectiveness. When a necessity for a computer-assisted instruction system becomes apparent, three major tasks must be performed: 1) development of computer software - in this case accomplished by ETS, 2) development of educational software - in this case accomplished by GIZMO, and 3) selection of teaching strategies - a yet-to-be-accomplished task assigned to lesson-writers made up of teachers and students[Stolurow, 1968].

II. GENERAL INFORMATION ON GIZMO

A. Execution Modes

The GIZMO system is basically comprised of three functionally independent programs. Teacher mode is used by authors of lessons to create their lessons. Teachers engage in a dynamic conversation with the program through a teletype. One of the most obvious tasks of this mode is to understand the teachers' requests and responses, render assistance as to what is needed at what time, and prevent any possible errors in lesson structure. A typical step of the dialog includes the teacher mode asking the teacher for a specific type of information (e.g., How many tries does the teacher want the student to have?), the teacher responding (e.g., 5), and the teacher mode analyzing the response and subsequently accepting or rejecting it. The output of this execution mode is a file containing the source lesson (see below).

The lesson compiler is a non-interactive program used by the teacher to translate the source lesson into the object lesson which is more easily interpretable by the student mode.

The student mode of GIZMO is the interactive program where a student "takes" the lesson that the teacher has previously created using teacher mode and compiled using the lesson compiler mode. A typical step in the dialog includes student mode printing text (optional), asking a question (e.g., How much is 2 plus 2?), the student responding (e.g., 4),

and the student mode analyzing and subsequently releasing pertinent information of concern to the student (e.g., Correct). See diagram below:

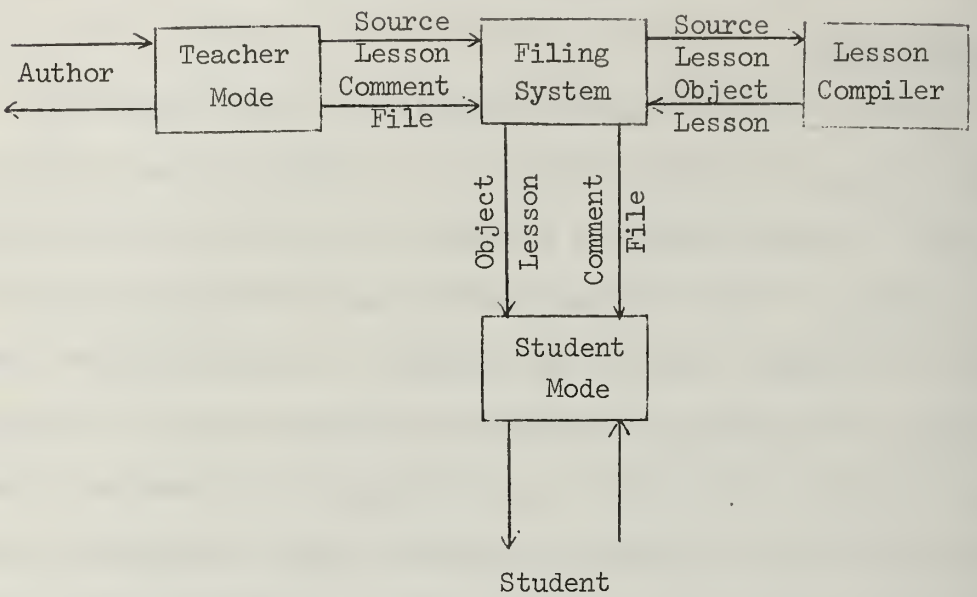


Figure II.1 GIZMO Structure - External

B. Lesson Modules

During the life of a lesson, there exist three modules, or lesson forms. The source lesson module is the initial lesson format. It is in a form very similar to that of the input lesson information supplied by the teacher or author. This lesson contains all necessary parameters and information to be eventually supplied to the student mode in the final form of the object lesson module. Because this is the lesson module that would be edited or altered if such a change were necessary, it must be both easily readable and editable. See appendix C for a sample of a typical source lesson module format.

The source lesson as described above is created by the teacher mode. At this same time, an additional file called the comment module is also produced. This comment module contains all of the text messages that GIZMO might eventually print out to the student in response to various answers. It also maintains a directory of these responses with pointers to their exact location in the module for fast and easy access. This comment module is passed on as is to the student mode. See appendix E.

The lesson compiler's task is to input the easily read source lesson into a more easily interpreted object lesson. Among other things it has but the source lesson does not, are values of Backus-Naur form expressions substituted for their names. Thus the GIZMO student mode need not have to search for a value; it already has it because of the work of the compiler. See appendix D.

C. Communication with Operating System

All three GIZMO programs run as system programs under Educational Timesharing System (ETS) on a PDP-11. GIZMO uses ETS's filing system and input/output facilities. I/O and filing system routines within GIZMO that serve as an interface with ETS are functionally isolated and easily replaceable with others to enable GIZMO to run under other PDP-11 operating systems. For a full explanation of ETS's filing system and I/O conventions see the references[Oxley, 1972].

III. GIZMO - AS VIEWED BY THE TEACHER

A. Basic Goals

As described in the introduction, one of the basic design criteria used in GIZMO is simplicity of lesson writing. The teacher or lesson-writer should be able to sit down at a console and be able to write a fair to good lesson without first having to read many pages of explanation of a new language. A teacher's manual has been created as an aid to the lesson author but is not absolutely essential. This manual has been reprinted in the appendices for the convenience of the reader and as a complete description of the language.

B. Achievement of Basic Goals

One of the traditional problems plaguing present CAI systems is the existence of a new higher level language used to write lessons which may be difficult for a non-computer-science teacher to learn. The author language for GIZMO is a very simple Backus-Naur form-like language which can be learned in its entirety in a matter of minutes. The essential constituent symbols to know are: < >, symbol delimiters; ::=, meaning "is assigned the value"; *, meaning "arbitrary number of occurrences of the previous symbol"; and |, meaning "or". One additional convention is that symbols whose names start with letters correspond to responses made by a student in answer to a question, and symbols whose names start with digits correspond to replies which the teacher wants GIZMO to give the student in response to particular student answer.

As a very crude example that employs all of the above concepts, suppose a teacher, when asked, in teacher mode, to give a correct answer, writes:

<INTEGR> <1>

which means, in english, if the student writes anything which corresponds to the definition of <INTEGR> then GIZMO should reply to the student by printing message number 1. Later GIZMO will ask the teacher to define both of these symbols if they are new to this lesson. When this happens, the teacher could write (here, as in all future examples, underlined text indicates computer-output and non-underlined text indicates user-input):

<INTEGR> ::= <DIGIT>*
<1> ::= THAT'S VERY GOOD.
<DIGIT> ::= 0|1|2|3|4|5|6|7|8|9

After these definitions are inserted, the teacher may use <INTEGR> at any time during this lesson to indicate an arbitrarily long string of the digits 0 through 9.

Many CAI systems are difficult for the author to use because he must define his own lesson structure. The lesson structure of GIZMO is "GIZMO-directed" not "teacher-directed." Instead of the teacher telling GIZMO what he wants to give next, GIZMO tells the teacher what it wants to receive next in his choice of format: either a detailed statement of what is required or just a one-letter code. The former is aimed obviously toward the beginning lesson-writer; the latter toward the more advanced one. The choice of operation is indicated by an option as described in the next section.

A GIZMO lesson has the following structure:

I. Title of lesson and options chosen

II. One or more questions, each with the following format:

1. Question number and question to student
2. Number of tries permitted
3. Correct answers
4. Incorrect answers
5. Response after number of tries exhausted
6. "GIVE-UP" response (only if option G for GIVE-UP specified)
7. "HELP" responses (only if option H for HELP specified)
8. Definitions of all new symbols

C. Writing GIZMO Lessons

We start with the example of figure III-1, in which the author of the lesson chose only three of the many options available. I shall explain below only the ones that he chose and a few others. For a complete description of the options available see the Teacher's Manual. The first option chosen here was "G". This indicates to GIZMO that the teacher wants "GIVE-UP" to be a reserved answer by the student and that the student should only type it when in fact he gives up. It also signals teacher mode to ask the teacher after each question what terminal remarks he wants printed in case the student types "GIVE-UP."

The second option specified was "H". This indicates "HELP" for the student and is similar to the G option except that it enables the teacher to render assistance to a troubled student and then permit him to try again.

```
GIZMO TEACHER MODE VER.IV 6/72
ID: T*0201
LESSON: EXMPL1
OPTION: G
OPTION: H
OPTION: S
OPTION:
QUESTION # AND QUESTION
1. PLEASE WRITE ANY ODD INTEGER IN BASE 10.
HOW MANY TRIES? 3
GIVE A RIGHT ANSWER
<INTEGR><ODDDIG><1>
GIVE A RIGHT ANSWER

GIVE A WRONG ANSWER
<INTEGR><2>
GIVE A WRONG ANSWER
<3>
GIVE A WRONG ANSWER

RESPONSE AFTER 3 INCORRECT RESPONSES.
<4>
RESPONSE FOR STUDENT'S "HELP"
<5>
RESPONSE FOR STUDENT'S "HELP"
<6>
RESPONSE FOR STUDENT'S "HELP"

RESPONSE FOR STUDENT'S "GIVE UP"
<4>
<INTEGR>::=<DIGITS>*↑
<ODDDIG>::=1↑3↑5↑7↑9
<1>::=VERY GOOD.  THAT IS INDEED AN ODD INTEGER.
<2>::=THAT IS AN INTEGER BUT ITS NOT ODD.
<3>::=THAT'S NOT EVEN AN INTEGER!
<4>::=YOU OBVIOUSLY DON'T KNOW TOO MUCH ABOUT THE INTEGER NUMBERS.
HERE ARE SOME EXAMPLES OF SOME ODD ONES: 3,6667,11111,987654321.
<5>::=AN ODD INTEGER ENDS WITH A 1,3,5,7, OR 9.
<6>::=TRY, FOR EXAMPLE, 73523.
<DIGITS>::=0↑2↑4↑6↑8↑<ODDDIG>
QUESTION # AND QUESTION
      .
      .
      .
```

Figure III.1 Example of Teacher Mode Dialogue

The third option specified was "S ". This indicates that the teacher wants to scrunch blanks out of the student's answer. That is, the teacher wants the student response 1 2 3 to be equivalent to 123, for example.

Another option that could have been chosen, but wasn't, is "Q". This stands for quick operation. It would suppress long requests in teacher mode such as QUESTION NUMBER AND QUESTION: which is replaced by Q:.

The teacher next typed the question to be given to the student. Note that this was done when GIZMO asked for it, not necessarily when the teacher was ready to give it--although the teacher had better be ready when GIZMO is! Next, the number of permissible tries was inserted as 3. These 3 tries include all entries by a student except the HELP's. A student is not penalized for asking for HELP.

When the teacher was asked for a correct answer, he wrote down his definition of an odd integer. Namely, any integer (<INTEGR>) terminated by an odd digit (<ODDDIG>). In the event that the student does type something matching this, then GIZMO has been directed by the teacher to print message <1> to the student. Note that <INTEGR> was later defined to be either an arbitrarily long string of digits or else the null string. Similarly <DIGITS> was defined to be any even or odd digit. I feel that the remainder of this excerpt is self-explanatory if one remembers, once again, that bracketed symbol names starting with a letter correspond to student responses and those starting with a digit correspond to teacher-directed and student-triggerred, GIZMO student mode responses.

Some other rules of lesson writing that may be handy to know are:

1. In specifying right and wrong answers, actual student responses may be included as well as bracketed symbols. Thus, for example A<INTEGR>. could be used to specify any answer which started with the letter A, was followed by any integer and was followed by a period.

2. A definition of a symbol starting with a letter may include actual student responses and additional bracketed symbols as necessary as well as |'s and *'s signifying or and repetition respectively.

3. A definition of a symbol starting with a digit may include additional bracketed symbols (whose names also start with a digit) as well as actual teacher response text.

4. GIZMO teacher mode will continue asking for more information of the same type when it is reasonable to assume multiple answers. For example, correct answers, incorrect answers, and responses to repeated student use of HELP all could have multiple responses. To indicate that you, the lesson author, have completed one of these sections, make a null entry and GIZMO will go on to the next section. A null entry is made by just typing a carriage return.

5. If GIZMO teacher mode notices that an entry is ambiguous, ill-defined, non-sensical or in any other sense unacceptable, the entry will not be accepted and GIZMO will say, "WHAT?."

In this chapter, I have not attempted to give a complete or formal description of how a teacher would write a lesson. I have simply tried to demonstrate the solutions that GIZMO has used to overcome two problems in lesson writing, namely, a new and usually complicated language and a teacher defined structure. By initially limiting GIZMO to a specific subset of

tasks, namely, drill and practice, the lesson-writing language can be kept very simple and the resulting lessons are observed to have few errors attributable to poor lesson structure or to poor knowledge of the language.

Extending my discussion to formal grammar theory, it is apparent that non-terminal symbols are in fact what I have been calling bracketed symbols; and that terminal symbols are in fact those characters inserted at any time by the lesson author during his specification of correct or incorrect answers or definition of non-terminal (bracketed) symbols. To find the language generated by GIZMO, let us apply the following procedure:

Step 1: For each correct or incorrect answer specified, generate the productions

$$S \rightarrow C_i$$

$$C_i \rightarrow \psi$$

or,

$$S \rightarrow I_i$$

$$I_i \rightarrow \psi$$

respectively where i is the number of the correct or incorrect answer and ψ is the string of terminal and/or non-terminal symbols of the answer.

Step 2: For the definition of each non-terminal symbol, A, that does not contain "or" signs, generate the production

$$A \rightarrow \psi$$

where ψ is the string of terminal and/or non-terminal symbols indicated by the definition.

Step 3: For each definition of a non-terminal symbol, A, that contains "or" signs, generate a set of productions

$$A \rightarrow \alpha_i$$

where the α_i's are the strings separated by the "or" signs.

Now that we have a set of productions of the GIZMO grammar, let me demonstrate that this grammar is a regular (type 3) grammar, and conversely, all regular languages can be described by GIZMO. Apply the following procedure to the previously developed grammar:

Step 1: Keep every production of the form

$$A \rightarrow \alpha$$

where α is a string of terminal symbols only.

Step 2: For each occurrence of a non-terminal symbol, A, that is followed by an * (for repetition) substitute it with a new non-terminal symbol, A^* and introduce the production

$$A^* \rightarrow AA^*$$

Step 3: All of the remaining productions contain at least one non-terminal symbol on the right side. Replace each of these with a set of productions in which the un-asterisked non-terminal symbols, B, are replaced by each and every one of its possible replacements as indicated by those other productions of the form

$$B \rightarrow \psi$$

Since these new productions might still contain non-terminals (because ψ might contain non-terminals) repeat step 2. This must be a finite process because GIZMO does not permit recursively defined bracketed symbols and because we have not included the *-ed symbols in this step. Notice that we have now removed all non-terminals from the right side of the productions except for those that were *-ed.

Step 4: For each production of the form

$$A \rightarrow \alpha C^*_{\psi}$$

where α is a string of terminals (possibly null) and ψ is a string of terminal and/or non-terminals (possibly null), remove that production but include the productions

$$A \rightarrow \alpha N_{C^*}$$
$$N_{C^*} \rightarrow CN_{C^*}$$
$$N_{C^*} \rightarrow \psi$$

If any change of this kind was made, go to step 3 in order to remove the non-terminal symbol C and those in ψ from the right side. Now all productions are in the form

$$A \rightarrow \alpha$$

or,

$$A \rightarrow \alpha A$$

where α is a string of terminal symbols. Thus the GIZMO author language does accept type 3 languages.

The converse is trivially true. For all productions of the form

$$A \rightarrow \alpha$$

supply α as the definition of <A>. For all productions of the form

$$A \rightarrow \alpha B$$

supply α<B> as the definition of <A>. For all productions of the form

$$A \rightarrow \alpha A$$

change all occurrences of <A> in a definition into <α>*<A>. If more than one definition for any bracketed symbol results, include them all in one definition with each of them separated by "or" signs.

IV. GIZMO - AS VIEWED BY THE STUDENT

To use GIZMO, a student performs a simple log-in procedure at a console and selects a lesson title from the posted list of lessons currently available. Then the computer presents text and questions for the student to study and answer. At any time, the student may type HELP which under lesson control causes the computer to supply the student with some hints, a restatement of the problem, a description of a possible approach to the problem, or anything else that the lesson author has suggested. Since a student is usually limited to a specific number of attempts at a particular question, he will not be penalized in this respect for asking for help. However, the teacher or lesson author can also specify a variety of HELP responses which when exhausted will result in the response, NO MORE HELP AVAILABLE.

If a student attempts to answer a question incorrectly, he will be informed by a lesson, if properly written, which parts of his answer are correct and which parts are not, with explanations of why they were wrong. He is then asked to try again. A student is usually permitted a specified number of attempts to answer each question. After exhausting this limit, the student will usually be given a different message which would either refer the student to certain books or chapters or else attempt to give the student a full explanation of the problem and its answer. This choice, of course, is up to the lesson author. At any time, a student may type GIVE-UP. This indicates to GIZMO that the student is not

interested in answering the question or is unable to do so. At this point the student is given another message specified by the author of the lesson.

The GIZMO authoring system has not been tried out by a large number of authors--as a matter of fact, I am the only one who has written any lessons so far.

Because GIZMO has been developed as an aid to teaching CS 201, an introductory course in assembly language and systems programming, I'd like to present here what I feel would be a good first iteration of lessons for this purpose:

Lesson #1: An introduction to computer and teletype usage, including practice performing simple tasks on the teletype such as finding specific keys (carriage return, line feed, rubout, etc.), and practicing typing messages using these keys. To be given during first week of the course.

Lesson #2: Base conversion. A review of base conversion including practice converting between any two bases from among binary, octal, decimal and hexadecimal. To be given during 2nd week of the course.

Lesson #3: Binary and octal arithmetic. Many examples and practice performing these types of arithmetic with emphasis on the results as they are indicated by the processor status word on the PDP-11 or IBM 360/75. To be given during 2nd week of the course.

Lessons #3 and #4: Introduction to basic assembly language concepts. Many examples and practice programming simple tasks as performed on trivial 0, 1, 2 and 3-address machines. To be given during 3rd week of the course.

Lesson #5: Introduction to 2-address instructions on the PDP-11. To be given during 4th week of the course.

Lesson #6: Introduction to 1-address instructions on the PDP-11. To be given during 4th week of the course.

Lessons #7 and #8: Introduction to addressing modes on the PDP-11. To be given during 5th week of the course.

Lesson #9: Summary. A review of the first 8 lessons plus an introduction to branch instructions. To be given during 6th week of the course.

Lesson #10: Exam. A GIZMO-administered, short answer, one-hour quiz on the past 6 weeks of material presented in lecture as well as in GIZMO lessons. To be administered during the second half of the 6th week.

I believe that this six week sequence of lessons (two lessons per week, approximately one hour per lesson) supplementing 3 hours of lecture and 1-2 hours of discussion would be an ideal approach to the problem of teaching students the large amount of material that is necessary for them to know before they can write their own programs.

CS 201 students should understand all of the basic material and concepts at the 1/3 point in the semester, before advancing to bigger and better things.

In summary, I would like to say that I think GIZMO can be a very effective aid to the students provided more staff become interested in the educational aspects of GIZMO and provided the students, as users of GIZMO, inform the staff of problems and advantages that they perceive in it.

V. INTERNAL STRUCTURE OF GIZMO

A. Teacher Mode (Author Mode)

Teacher mode of GIZMO is basically comprised of 7 parts: user validator, message coordinator, message printer and response reader, general response scanner, particular response checkers, symbol table handler, and ETS interface routines. Figure V-1 demonstrates the general hierarchy or flow of control between these various parts.

Let me now explain what tasks each of these routines are assigned and in general terms, how each one performs its tasks.

The user validator routine is designed to prevent anybody except specific recognized authors from obtaining access to the functional parts of teacher mode and therefore be capable of writing lessons, and prevent anybody except the original author from modifying a previously written lesson. All of these tasks are performed easily with a six character code word assigned to each lesson and each author. At present, only one code word is valid and is being used by all authors. Additional code words can be added easily later. This routine also creates and opens appropriate files using the ETS interface programs.

The message coordinator routine's main task is to keep track of messages (excluding error messages) which should be given to the teacher. These messages are the cues given the teacher in order to prevent lesson errors due to poor lesson structure. This subroutine also has other tasks

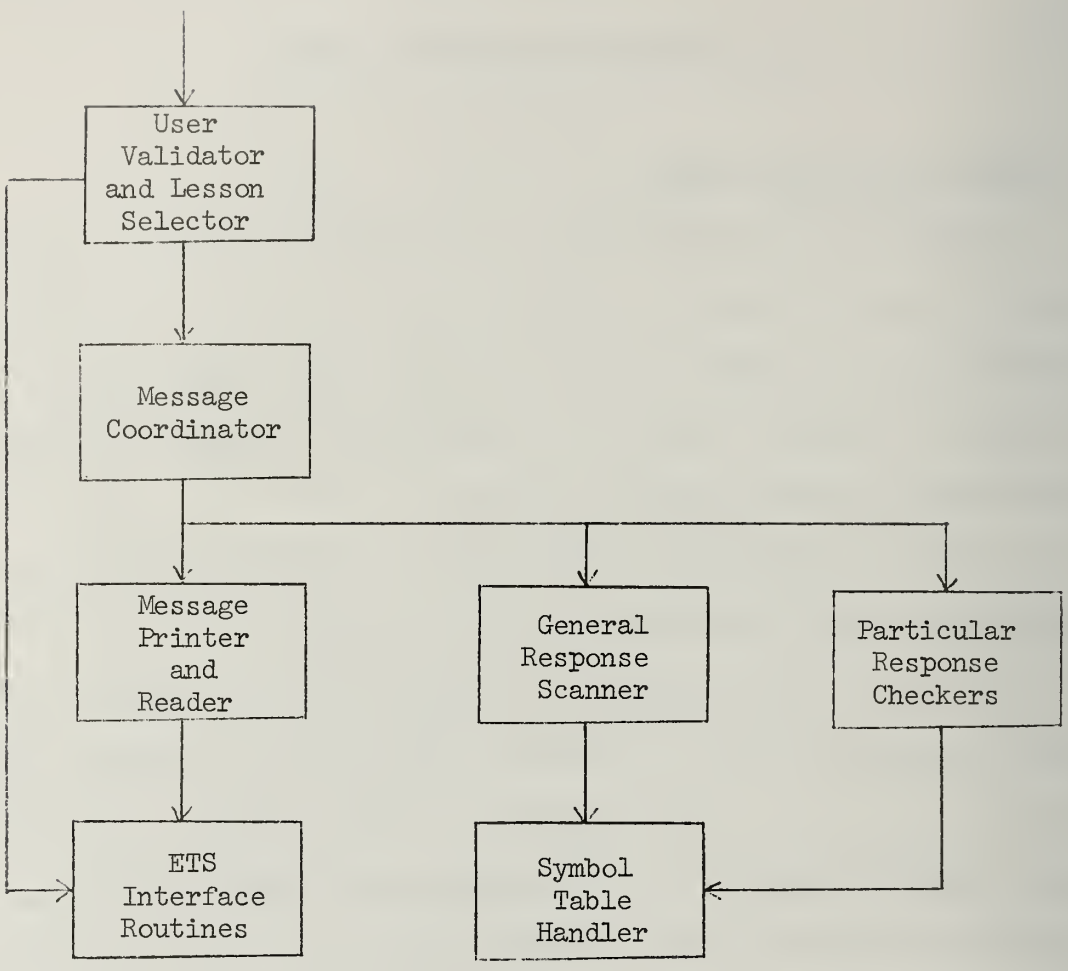


Figure V.1 Teacher Mode Hierarchy

including calling the routines which print the cues, scan for general syntax errors, and scan for specific errors possible within certain kinds of responses. For more details of these routines, see their descriptions that follow.

The message printer and reader gives the instruction to the teacher that its calling program requests and then accepts the teacher's response.

The general response scanner (called T.SCAN in the actual code) performs a general syntax check and looks for obvious errors. Among the possible errors that it could find are unmatched brackets, illegal characters and badly formed symbol names. It also inserts new symbols in the symbol table.

The particular response checkers check number and type of parameters in options, check for null responses when important, check for a valid number in the case of number of tries, etc. Both these and T.SCAN have the right to terminate examination and processing of any response and cause "WHAT?" printed to the teacher.

The symbol table handler (called T.HASH in the actual code) keeps track of all symbols used by the teacher in the present lesson. It knows whether any symbol has been used and whether or not it has been defined. After each question and answer set has been generated, teacher mode, using this routine, asks the teacher to define any new symbols used that are not already defined.

The ETS interface routines enable GIZMO to communicate with the outside world and the operating system. These are the same in all three GIZMO modes and include the following tasks:

1. File Handling

    a. Create a file X.

b. Open a file X on I/O channel Y.

c. Close channel Y.

2. Input/Output

a. Transfer block W of file X to core locations Z.

b. Transfer core locations Z to block W of file X.

c. Read a line from the teletype.

d. Print a line on the teletype from core location Z.

e. Print a line on the teletype from block W of file X.

B. Lesson Compiler

The teacher mode outputs a source lesson very similar to the actual input supplied by the teacher. If this lesson were actually used during student mode much time would be spent looking for definitions of symbols. The lesson compiler alleviates part of this overhead by doing an expansion of symbol names into symbol values throughout the lesson. The compiler has two passes. Pass #1 simply constructs a symbol table containing the location of the definitions of all symbols used in the lesson. Pass #2 makes use of this table to expand the lesson. See figure V.2.

Pass #1 does a simple linear search through the entire lesson for all occurrences of the string "<symbol name>::=". It then records in the symbol table: the symbol's name, the block number in the lesson file where its definition can be found, and the offset in bytes from the beginning of that block in order to be able to locate its precise location easily.

Pass #2 also performs a linear search of the source lesson for all occurrences of a symbol name in brackets. When found, it transfers control to the symbol handler (called G.TABS in the actual code) which locates the

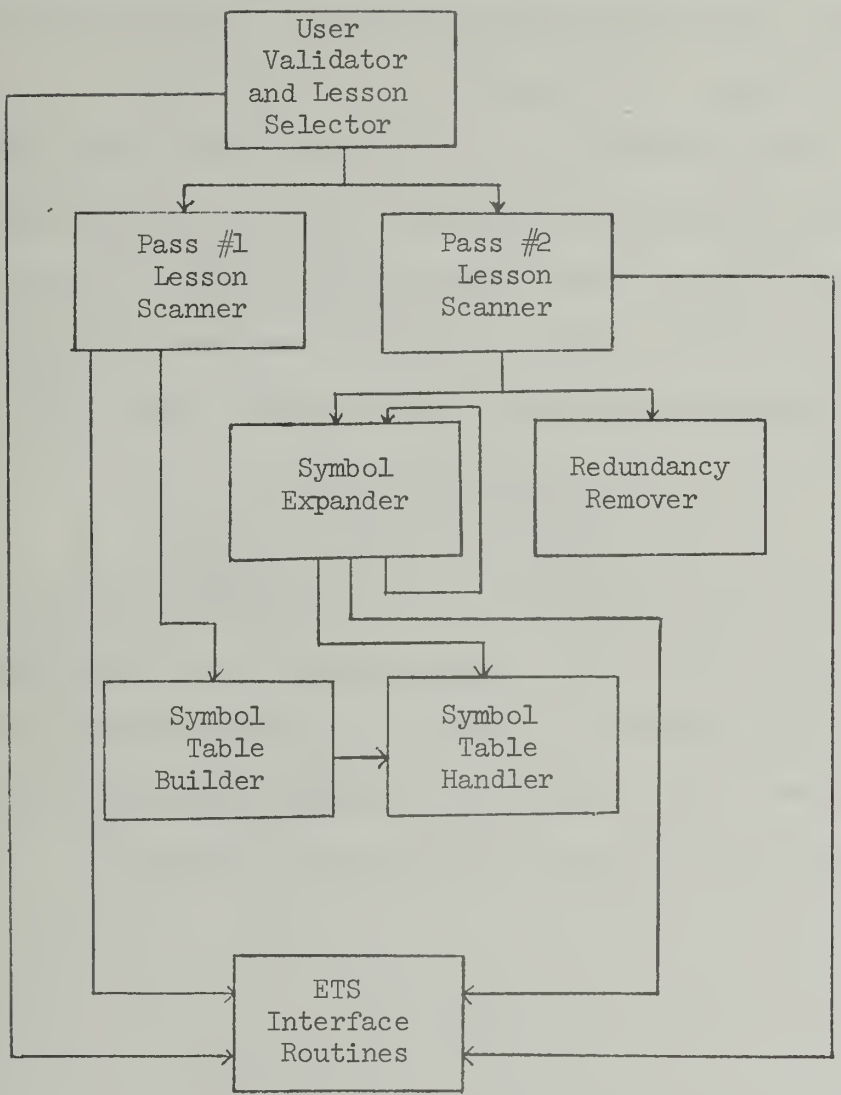


Figure V.2 Lesson Compiler Hierarchy

value of the symbol and replaces the name of the symbol with the value. However, while doing this, it often finds another bracketed symbol name. In this case, G.TABS calls itself recursively and then continues on processing the line. See figure V.3.

The lesson compiler also has a routine which removes redundant brackets from the expanded definitions. Since all bracketed labels are removed from the lesson during compilation, the object lesson uses brackets as logical parentheses. These are generated around any string or set of strings that had previously been a single bracketed label. For example, if

<SYMBOL>::=X|Y|Z

then in place of each occurrence of the string <SYMBOL> in the lesson would be compiled the string <X|Y|Z>. This maintains the lesson-author's logic but may lead to redundancy of brackets. The lesson compiler's redundant brackets remover (called REDUND in the actual code) removes the indicated (by ↑) brackets when they occur in the following contexts:

1. ... << ... >> ...
   ↑ ↑

2. ... < ... |< ... | ... >> ...
   ↑ ↑

3. ... << ... | ... >| ... > ...
   ↑ ↑

The author is aware that other conditions can exist where brackets could be safely removed without destroying the lesson-author's original intent. However at this stage of development, it was noticed that these were the most common and easiest to locate and therefore included. Other types will be removed after further analysis has been completed.

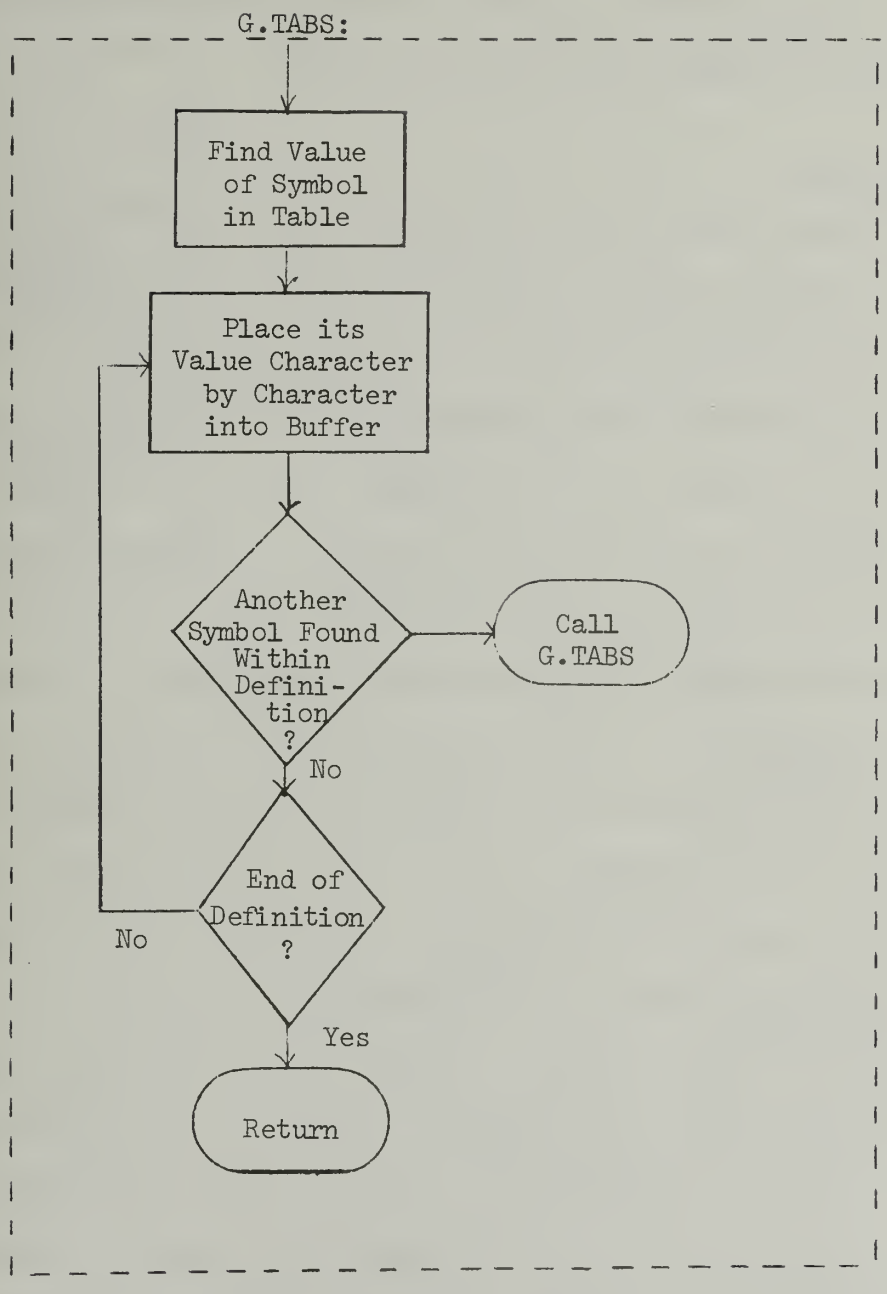


Figure V.3 Logic of Symbol Processor in Lesson Compiler

The lesson compiler uses the identical symbol table handler and ETS interface subroutines that the teacher uses.

C. Student Mode

In simple terms, the student mode's task is to print questions to the student, read his answer, evaluate its validity, return appropriate messages and then determine whether to repeat this same question or go on to the next. These tasks are performed within the framework of figure V.4.

The main program's job (called STMODE in the actual code) is to start the interpreting process on the lesson. It examines all the options chosen and records these in easily accessible tables. For example, all characters to be scrunched (option S) and all characters to be ignored (option I) are recorded in the SCRIGN table to be used later by other routines. This main program also sequences through the questions of the lesson one by one and transmits control to the question printer, answer reader, and the lesson part coordinator. In general, the main program keeps track of which question is presently being processed.

The lesson part coordinator (called TABLE in the actual code) coordinates the analysis of the student's response. It keeps track of whether the comparer is comparing the response with an answer which the teacher has indicated to be a correct or incorrect answer, or whether the student's response was a reserved answer (e.g., GIVE-UP or HELP if option G and/or H were specified). During the answer matching, TABLE also records and collects the numbers of all the responses that might be printed to the student at the conclusion of the analysis.

The comparer (called LOOKER in the actual code) of GIZMO performs an initial call to the main workhouse of the analysis process, the character

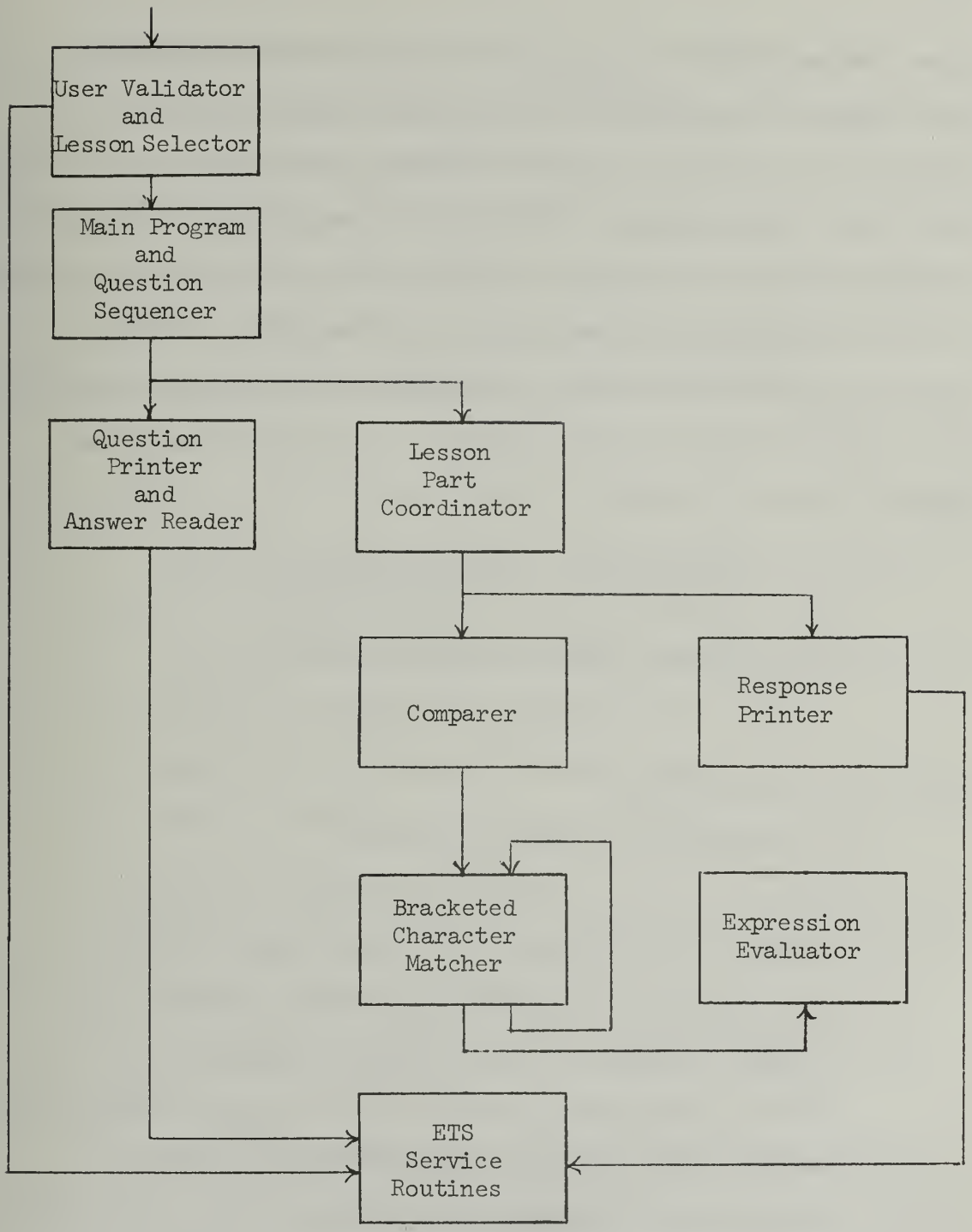


Figure V.4 Student Mode Hierarchy

matcher (called MATCHER in the actual code). When MATCHER is called, he is given a pointer to a position in the student's answer and a pointer to somewhere in the string of characters defining the possible answers predicted by the author of the lesson. MATCHER uses a simple backtracking method of scanning a teacher's answer. To demonstrate this, let the predicted answer be of the form $P_1P_2\ldots P_m$ where each part P_i is comprised of a set of n strings $\{S_{i_j} \mid j=1\rightarrow n\}$ separated by "or" symbols. Thus the answer specified by the teacher is (that is, after expansion of symbols by the lesson compiler):

$$<S_{11}|S_{12}|\ldots|S_{1n}><S_{21}|S_{22}|\ldots|S_{2n}>\ldots<S_{m1}|S_{m2}|\ldots|S_{mn}>$$

which would mean that any student response of the form

$$S_{1i_1}S_{2i_2}\ldots S_{mi_m}$$

matches the specified answer.

The backtracking scan works as follows:

Algorithm A: This is a backtracking technique which will attempt to parse a particular student answer S. In the algorithm, a counter I keeps track of which of the m pairs of brackets we are presently scanning within; PART (I) keeps track of which of the n strings within bracket pair I we are presently examining; POS (I) keeps track of position within student's response when bracket pair I is encountered.

Step A1: [Initialization] I←1; PART (I)←1; POS (I)←pointer to 1st character in student's answer.

Step A2: [Find string?] If $S_{I,PART(I)}$ = substring of student's answer starting at POS (I) then go to step A5, else PART (I)←PART (I) + 1.

Step A3: [Try next string or else backtrack] If PART (I) ≠ n + 1 then go to step A2, else I ← I - 1.

Step A4: [Backtrack] If I = 0 then stop algorithm and NOMATCH, else PART (I) ← PART (I) + 1 and go to step A3.

Step A5: [Match so far] POS (I+1) = POS (I) + length of string $S_{I,PART(I)}$; I ← I + 1; PART (I)←1.

Step A6: [Complete match?] If I = m + 1 and POS (I) = end of student's answer, then stop algorithm and MATCH. If I = m + 1 and POS (I) ≠ end of student's answer, then PART (I)←PART (I) + 1 and go to step A3. If I ≠ m + 1 then go to step A2.

Actually, the algorithm used by GIZMO is more complex than this due to the following additional problems that may occur:

1. The bracketed clauses may have a variable number of ored substrings.

2. Embedded anywhere in the teacher's answer may be comment numbers (i.e., labels beginning with numbers) to be recorded for later release to the student.

3. Specific characters in the student's response must be edited (removed from their response) or ignored (considered as a response terminator) if options S, for scrunch, or I, for ignore, were chosen.

4. Specific student responses must be handled differently. For example, the responses of "HELP" or "GIVE-UP", if option H or G were chosen.

5. Certain character strings in the teacher's answer must be considered differently. For example, if option E, for evaluate, was chosen, then all numbers in the teacher's answer must produce a call to the expression evaluator to evaluate the present part of the student's response to see if it is actually an arbitrarily complex arithmetic expression whose value equals the number specified by the teacher.

6. Actual teacher's answers might appear without brackets surrounding them.

The expression evaluator (called EVALO in the actual code) is induced provided both of these two conditions are present:

1. Option E had been specified by the lesson author in the lesson heading, and

2. An integer is discovered by the scanner of the teacher's answer.

When both of these conditions are met EVALO evaluates the arithmetic expression at the present cursor position in the student's answer and returns its value. Of course, if no expression exists here, failure to match at this point is declared and the backtracking algorithm takes over again.

In the event that GIZMO is successful in producing a complete parse of a student's answer (i.e., matches one teacher's answer completely) only that collection of comments encountered during that particular parse are printed to the student. In the event that GIZMO is unsuccessful, all comments encountered during all partially successful parses are printed to the student. In either case, these sets of messages are printed by the response printer (called CMPRNT in the actual code). CMPRNT is the only

routine in all of GIZMO which requires access to the comment file (one of the 3 lesson modules) after its creation during student mode. By accessing the table at the beginning of this file, CMPRNT is capable of locating any message quickly and easily. For information concerning the structure of this module, see appendix E.

Student mode uses the identical ETS-interface subroutines that teacher mode and the lesson compiler use.

In this chapter, I have attempted to give a full explanation of the top-down structure of the three modes of GIZMO. I have only included certain algorithms used during the bottom-up development where I felt these were particularly interesting, unique, or else necessary for the reader to understand in order to gain a more thorough knowledge of how GIZMO actually works.

VI. ANALYSIS AND EVALUATION

A. CAI Systems

When a CAI system is developed, there are certain responsibilities which it must accept. Among the essential services are: 1) The CAI system must be able to evaluate a student's response by comparing it to various answers predicted by the teacher. 2) The system should provide the teacher with detailed data on student performance (However, some CAI experts claim that teachers don't know how to evaluate this data yet! [Rogers, 1968]). 3) The system should make it easy for a lesson author to both create and modify the lesson. These three services could be used to define a CAI system if definition by property is sufficient. These are a subset of the factors presented by Zinn when talking about what factors affect the value of a CAI system[Zinn, 1968].

Computer assisted instruction also has some major advantages over human teaching. CAI terminals provide individual instruction to a large number of students at once. While a human teacher can lecture only to a large group, a properly constructed computer system could supply each student with a lesson which can adjust to the individual's rate of learning automatically. The existence of a CAI system also provides educators and educational researchers with a medium for exactly reproducible lesson presentation, complete accumulation and analysis of student performance data, research on teaching methods and development of instructional materials. In

addition, as the CAI system absorbs more and more of the "menial" jobs of presentation of material, checking for stragglers in a class, drill and practice and examination, the human teacher is finally in a position to perform tasks more human. The teacher can devote his or her time to such activities as group discussions where the students can learn different kinds of material not applicable to presentation by a computer[Stolurow, 1968; Hansen, 1970].

Hansen and Harvey[1970] do an excellent job of analyzing the changing roles of teachers as CAI's are introduced. In brief they say: 1) teachers will be doing less presentational functions, 2) teachers will be performing less corrective and negative verbal behavior, 3) teachers will become more concerned with teaching strategies and thus increase general teaching effectiveness, 4) teachers will become more involved in individual guidance, 5) teachers will become more involved in discussions with students rather than in lectures to them, and 6) teaching will become a team project for a group of professionals including systems programmers, specialists in guidance and specialists in lesson planning.

In addition to the impact upon the present teachers, CAI systems also should have an equally strong effect on present curriculum. Human teachers have somehow managed to be fairly successful at teaching even though they rarely have well defined objectives or organized material. CAI lessons require more organization and interest in details than human-presented lessons. Obviously, the ones to gain from this reestablishment of educational procedures are the students. Bravo! Computerized lessons require careful planning at lower levels as well as on the level of

structure and objectives. All too often the computer scientist doubling as a lesson writer could be unhappily detected by the student when he's faced with a response from the computer that reads, "ILLEGAL" rather than a slightly more human response of "PLEASE REPHASE," or "I DO NOT UNDERSTAND _____."

In the development of a CAI lesson writing language, one must consider that there are at least 5 different kinds of authors: instructors, authors, instructional researchers, administrators, and computer programmers. The personal attributes of each of these groups would suggest personal attributes to be included in the language. However, the existing languages do not fall into categories by user or lesson writer[Zinn, 1968].

B. CAI Languages

Various CAI experts have attempted to define CAI language criteria or CAI language classes in an attempt to contrast the capabilities and characteristics of various lesson writing languages. Notably among these are Karl Zinn[1967, 1968, 1970] and Charles Frye[1968]. I will extract from Frye's article as I personally feel that his classification procedure is superior to Zinn's. The four classes of languages are conventional compiler languages, adapted conventional compiler languages, interactive computing and display languages, and specially devised instructional author languages. Conventional compiler languages can only be used by an experienced programmer effectively. Languages such as FORTRAN, ALGOL and COBOL have been used to construct CAI lessons, but they're quite inefficient for coding many instructional tasks among which are matching answers (especially if a match that is only partially correct must be located) and maintaining student performance data.

Adapted compiler languages, like CATO (U. of I.) of FORTRAN, ELIZA (M.I.T.) of OPL and FOIL (U. of Michigan) of FORTRAN make it easier to perform specific tasks such as answer matching and student record management. There still, however, is the major problem of having all authors be programmers.

Interactive computing and display languages provide the additional capability of effective and fast response interaction between the student and computer. Such languages as APL, BASIC, and QUIKTRAN, however, do not give the author of the lesson service routines for answer matching, criterion branching or student record retrieval. The main advantage of these languages is that they can assist a student in his attempt to find solutions to certain problems. One way that these interactive languages perform this is in their ability to diagnose programming errors as they are typed in.

The specially designed author languages have an added bonus to the three previously mentioned classes. Not only can they find syntax errors, they can also find errors that might occur because of their instructional environment. For example, one of these languages could inform the author when he makes a mistake in entering lesson text while answer matching is not a normal function of conventional compiler languages and therefore they could not perform the equivalent. Languages such as COURSEWRITER (IBM), FOCAL (OISE), PLANIT (SDC), TUTOR (U. of I.) and GIZMO provide the lesson writer with easy to use answer matching facilities as well as facilities for building instructional sequences, monitoring student activities, and entering correct and

incorrect answers. Some also provide for misspellings (TUTOR), algebraically equivalent answers (PLANIT and GIZMO), and means for writing branching rules dependent on the history of the student's responses.

Zinn classifies author languages by: presentation of successive frames or items, conversation within a limited context, presentation of a curriculum file by a standard procedure, and interactive programming languages for student use. I do not feel that these group titles are clear enough or specific enough for my presentation here.

C. GIZMO

"If CAI is still so primitive and still has so many obvious disadvantages, why do we bother with it at all? I think the same question could have been asked at the birth of most new technologies. The first airplanes looked silly, too. It isn't what CAI is now that's important, its what it could become..."[Stansfield, 1972]. As with airplanes in their early years, CAI's are totally experimental at this point. I do not believe that at this point any group of people will state that they have found the absolutely best computer assisted instruction system. Each installation will produce certain aspects or features which were totally experimental at their beginning and will turn out to be excellent in retrospect. GIZMO was a highly experimental (two reasons for calling it experimental are that it was originally student designed and the author language is extremely simple) system and employs certain features which had to be tried before anybody could declare a totally successful CAI system. Stansfield[1972] advises that CAI's be kept at an experimental basis at the present time; that the present CAI systems are not yet ready for schools and schools are not yet ready for CAI.

Did GIZMO accomplish the tasks it set out to accomplish? Did it maintain the standards originally prescribed? As described in the introduction, the original goals were: 1) to provide a programming exercise for the CS 201 students, 2) to provide a system which can be easily understood and modified by future CS 201 students if necessary, and 3) to assist in teaching CS 201. The original GIZMO was, I must admit, a disaster as far as a program goes. The structure was clear and easy to understand but the code was awful (with a few notable exceptions). It required a major rewrite on the author's part to create a presentable program. Further attempts to permit CS 201 students to modify GIZMO proved equally disastrous. The author is now firmly convinced that students involved in a first semester introductory assembly language programming course should be required to plan, write, and debug many small to medium sized programs. Perhaps during a second more advanced course, they should be set loose on a large multi-programmer project that requires easily enforced communication between groups but difficultly enforced clean internal code. Thus, I might say that original goal #1 was accomplished only in part, in that the students did learn to plan and organize a large software project with all the usual problems of communication between teams present, but they also failed to gain another programming skill: basic, clean, uncluttered, easy to follow code. Original goal #2 also has been accomplished only in part. The system as is, I believe can be understood by most advanced CS 201 students because of the documentation, modularity and attempted simplicity. However, whether the author wishes to permit them to modify it again is doubtful. Original goal #3 has not yet been realized due to a major lack of interested lesson writers and repeatedly unsuccessful attempts to maintain teletypes in working order.

In light of the above, the goals of GIZMO have been changed: 1) to be a learning experience for the author, 2) to assist in the teaching of CS 201, 3) to establish an experimental CAI system with emphasis on simplicity (for the teacher, student, lesson, and program), and 4) to provide a foundation for further research and investigation into combining CAI systems and dynamic supervision of computer programming. Goal #1 above has been totally successful. Goal #2 still remains until the problems described above are alleviated. Goal #3 has been satisfied sufficiently, too. Goal #4 is the subject of the next chapter.

VII. DIRECTIONS

A. Using Present Version

Because of the internal modularity of the present version of GIZMO, it is quite easy to remove or include various options, capabilities and facilities. I would like to discuss a number of these that should be included during the next stage of GIZMO's development.

One of the basic necessities of an operational CAI system is the existence of a student data base (SDB). This file should contain: 1) vital statistics on all valid student users such as ID number, age, class, course, professor, etc., 2) past performance records of how each student has done on previous lessons and examinations and perhaps in previous courses as well as a list of problem areas that the student has encountered in the past, and, 3) present performance record of how the student is doing on the present lesson. In addition to this, the system should provide easy access to any of these types of information to the qualified personnel whose job it is to evaluate student performance.

Presently, questions are presented linearly to the students. When a student either answers a question correctly or else answers incorrectly a number of times, he is then given the next question. It is believed that this method maintains the students on the same level and does not provide individualization for the student who is excessively fast or slow. Thus, GIZMO should be given the feature that the next question or group of questions be a function of the STB, the performance on the just

completed question or group of questions, and the sequencing information supplied by the teacher. On another level there should be established a sequencing method for lessons. Max Jerman[1972] suggests 5 levels of competency for each lesson subject and that the student's level should be determined by his performance on a previous lesson (see figure VII.1). He used this method for teaching elementary mathematics at Stanford. I suggest that for teaching assembly language that only medium to high-level competency on any one subject matter should permit the student to proceed. If a student cannot perform well on the medium level drills, he will not be allowed to go on; he should spend additional hours per week taking review sessions and other lessons on the same material until this competency level is attained (see figure VII.2).

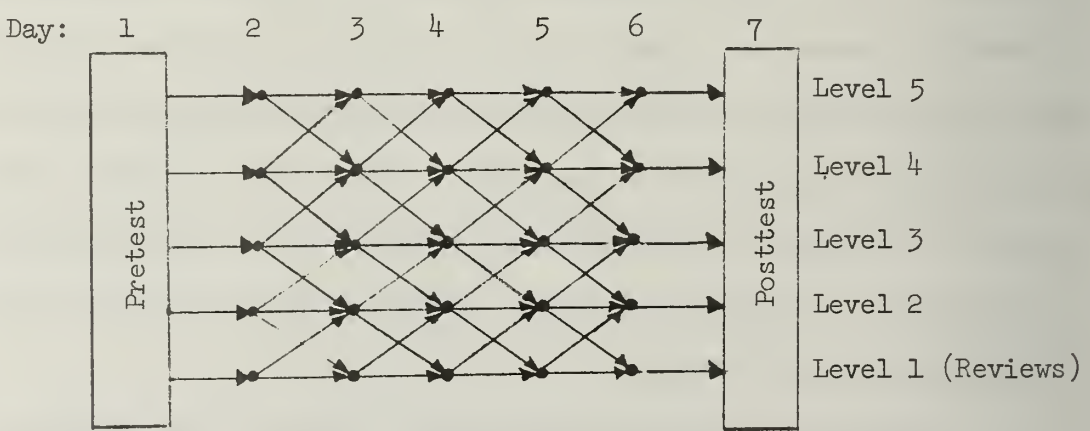


Figure VII.1 Max Jerman's Branching Structure for the Stanford Drill-and-Practice Program in Arithmetic

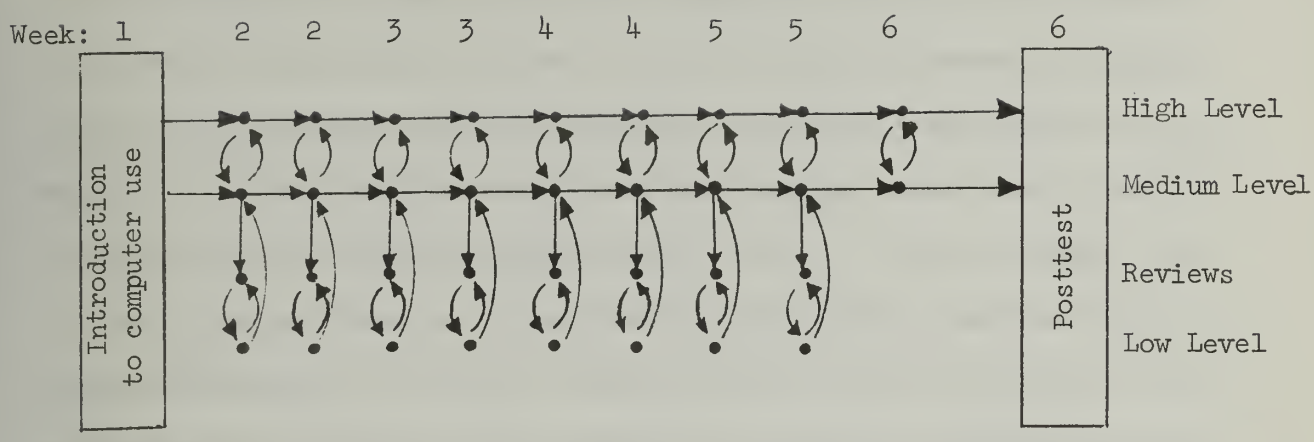


Figure VII.2 Proposed Branching Structure for First Third of CS 201 Drill-and-Practice on GIZMO

In addition to these changes in general use, there are also some additional properties I'd like to introduce into teacher mode. Among these are shortened forms of lesson structure that can be specified in the event that the entire lesson will be multiple-choice or true-false questions, a timer option to give students a limit on time of response as well as number of tries, and a random number generator implemented to produce random data for specific questions. Also to be added to teacher mode should be more error checking related to lesson logic as well as a lesson editor and automatic mode switcher. For example, an editor is required to permit a teacher or lesson author to add, delete or modify lesson parts. At present, an error can only be corrected by using the operating system's general character editor on the source lesson module. This is obviously quite dangerous since one can now affect the lesson in such a way that it no longer follows the correct syntax of a source lesson, and thus ruin its

possibilities of being compiled by the lesson compiler correctly. When I speak of an automatic mode switcher, I mean the capability of a lesson writer to stop in the middle of a partially written lesson, try out some of the questions as a student and then either edit or continue writing the lesson where he left off. This is an obvious asset and is not in the present version of GIZMO although does exist in most other CAI systems.

All of the above mentioned modifications can be implemented easily into the framework of the present GIZMO, and will, I hope, be implemented soon. I would now like to discuss some of the advanced CAI features that should be studied and perhaps implemented in later versions.

B. For Later Versions

There are four closely related facilities in this category: 1) linkage with external routines, 2) computer simulation, and 3) dynamic supervision of program writing.

If a lesson author wants to develop some special purpose algorithms to be used in answer matching, a CAI system should provide for this flexibility by permitting linkage of the author's external (external to GIZMO, that is) algorithm in assembly or machine language form. After linkage, GIZMO would transfer processor control to this other program which would evaluate the students' responses and return the proper information to GIZMO such as validity of answer and messages, if any, to give the student.

One area of application for the linkage of external program facility would be to provide computer simulation abilities. A previously written computer simulator could be used to simulate a task indicated by a student's program and compare the results with the previously defined results specified

by the lesson author. This would permit GIZMO to ask questions of the type, "Write a program which performs the following such and such tasks," and GIZMO could properly judge the results. This would give GIZMO a capability in teaching programming that no other CAI system that I know of could perform as quickly and efficiently.

A further extension of the idea of using CAI to teach assembly language programming lies in the realm of linking GIZMO with an incremental assembler and a very smart simulator with the resulting capability to evaluate the student program: 1) while it is being written for obvious syntax or simple logical or structural errors and 2) while it is being executed for further errors in logic or inefficiency and side-effects. In addition the facility could evaluate the student on size of source program, size of object program, speed of execution, as well as correct results--all using criteria and standards set down by the lesson author and in this case, he is probably also the instructor.

In summary, I think that there exists a vast area for expansion and correction in the area of GIZMO as well as CAI's in general. The possibilities of teaching programming in assembly language or higher level language using a process of dynamic analysis of the student's work are big.

LIST OF REFERENCES

Davis, Alan, et al., "Instructional Software for an Assembly Language Course Taught on a Mini-Computer," in William G. Bulgren, and Earl J. Schweppe, eds., Proceedings of the ACM SIGPLAN Symposium on Pedagogic Languages with Small Computers, Lawrence, Kansas: University of Kansas, 1972, pp. 236-248.

_____, et al., ETS Student's Guide, Urbana, Illinois: University of Illinois publication number UIUCDCS-R-72-547, 1973.

Diamond, Herbert S., "The Writing of a CAI Program by an Author New to Computers," Educational Technology, XI, 10 (October, 1971), 42.

Feingold, Samuel L., "PLANIT - A Language for CAI," Datamation, XIV, 9 (September, 1968), 41-47.

Frye, Charles H., "CAI Languages: Capabilities and Applications," Datamation, XIV, 9 (September, 1968), 34-37.

Hansen, Duncan N. and Harvey, William L., "The Impact of CAI on Classroom Teachers," Educational Technology, X, 2 (February, 1970), 46-48.

Hicks, B. L. and Hunka, S., The Teacher and the Computer, Philadelphia: W. B. Saunders Co., 1972.

Jerman, Max., "Promising Developments in Computer Assisted Instruction," in The Educational Technology Reviews Series, Englewood Cliffs, New Jersey: Educational Technology Publications, 1972.

Kopstein, Felix F., "Why CAI Must Fail!" in The Educational Technology Reviews Series, Englewood Cliffs, New Jersey: Educational Technology Publications, 1972.

Meredith, J. C., "Machine as Tutor," in The Educational Technology Reviews Series, Englewood Cliffs, New Jersey: Educational Technology Publications, 1972.

Oxley, Donald W., ETS System User's Guide, Urbana, Illinois: University of Illinois publication number UIUCDCS-R-72-523, 1972.

_____, The Design and Implementation of an Educational Time-Sharing System for the PDP-11, Urbana, Illinois: University of Illinois publication number UIUCDCS-R-72-520, 1972.

Rogers, James L., "Current Problems in CAI," Datamation, XIV, 9 (September, 1968), 28-33.

Salisbury, Alan B., "An Overview of CAI," Educational Technology, XI, 10 (October, 1971), 48-50.

Stansfield, David, "The Computer and Education," in The Educational Technology Reviews Series, Englewood Cliffs, New Jersey: Educational Technology Publications, 1972.

Stolurow, Lawrence, Computer Assisted Instruction, Detroit: American Data Processing, Inc., 1968.

Zinn, Karl L., "Computer Technology for Teaching and Research on Education," Review of Educational Research, 37 (1967), 618-634.

_____, "Instructional Uses of Interactive Computer Systems," Datamation, XIV, 9 (September, 1968), 22-27.

_____, "Programming Conversational Use of Computers for Instruction," Proceedings of the 23rd National Conference of the ACM, Princeton, New Jersey Brandon Systems Press, Inc., 1968, pp. 85-92.

_____, "Instructional Programming Languages," Educational Technology, XI, 3 (March, 1970), 43-46.

_____, "Implications of Programming Languages for Mathematics Instruction Using Computers," a paper given at a special meeting on CAI in mathematics education at the National Council of Teachers of Mathematics annual meeting, 1968, discussed in John F. Feldhusen and Michael Szabo, "A Review of Developments in Computer Assisted Instruction," in The Educational Technology Reviews Series, Englewood Cliffs, New Jersey: Educational Technology Publications, 1972.

APPENDIX A. SAMPLE DIALOGUE BETWEEN TEACHER AND TEACHER MODE AND COMPILER

```
GIZMO TEACHER MODE VER.IV 6/72
ID: T*0201
LESSON: EXMPL2
OPTION: S
OPTION: I;
OPTION: G
OPTION: H
OPTION: E
OPTION:
QUESTION # AND QUESTION
1. WRITE A PDP-11 ASSEMBLY LANGUAGE INSTRUCTION THAT ADDS THE
NUMBER ONE TO THE CONTENTS OF REGISTER 0.
HOW MANY TRIES? 4
GIVE A RIGHT ANSWER
ADD#1,<REG0><1><3>
GIVE A RIGHT ANSWER
SUB#177777,<REG0><12><3>
GIVE A RIGHT ANSWER
INC <REG0><2>
GIVE A RIGHT ANSWER
<WORD><10><12>
GIVE A RIGHT ANSWER

GIVE A WRONG ANSWER
<ADDER><ONER><COMMA><NOTREG><1>
GIVE A WRONG ANSWER
<SUBER><NONER><COMMA><NOTREG><1>
GIVE A WRONG ANSWER
<INCER><NOTREG>
GIVE A WRONG ANSWER
<SUBER><NOTREG><COMMA><NONER><1><4>
GIVE A WRONG ANSWER
<ADDER><NOTREG><COMMA><ONER><1><4>
GIVE A WRONG ANSWER

RESPONSE AFTER 4 INCORRECT RESPONSES.
<19>
RESPONSE FOR STUDENT'S "HELP"
<20>
RESPONSE FOR STUDENT'S "HELP"

RESPONSE FOR STUDENT'S "GIVE UP"
<19>
<REG0>:::=%0↑R0
<1>:::=THAT'S NOT VERY EFFICIENT.
<3>:::=INC R0  IS BETTER.  BUT YOURS IS O.K. TOO.
<12>:::=YOU'RE TRYING TO BE A WISE GUY.
<2>:::=VERY GOOD.
<WORD>:::=.WORD 5200↑5200<11>
<10>:::=O.K. BUT A PSEUDO-OP IS NOT REALLY A PROPER PAL INSTRUCTION.
<ADDER>:::=ADDB<5>↑ADD
```

<ONER>:::=#1↑1<6>
<COMMA>:::=,↑<7>
<NOTREG>:::=REG0<8>↑REG<8>↑R0↑0<8>
<SUBER>:::=SUBB<5>↑SUB
<NONER>:::=#177777↑177777<6>
<INCER>:::=INCB<9>↑INC
<4>:::=THIS IS NOT THE 360/. THE DESTINATION ADDRESS APPEARS ON THE RIGHT, NOT ON THE LEFT.
<19>:::=THE BEST ANSWER IS

```
        INC R0          ;ADD ONE TO REGISTER ZERO
```

BUT

```
        ADD #1,R0
```

ALSO WORKS, BUT IS LESS EFFICIENT.
<20>:::=FIND AN APPROPRIATE 1-ADDRESS INSTRUCTION FROM PAGE 92 IN THE PDP-11 HANDBOOK.
<11>:::=IT'S A DEFAULT PSEUDO-OP.
<5>:::=THAT OPERATOR IS NOT A VALID ONE. BETTER CHECK YOUR HANDBOOK.
<6>:::=ALL CONSTANTS(LITERALS) MUST BE PREFIXED BY A # SIGN.
<7>:::=THE OPERANDS IN A 2-ADDRESS INSTRUCTION ARE SEPERATED BY A COMMA. YOU LEFT IT OUT.
<8>:::=THAT IS NOT A VALID WAY TO EXPRESS REGISTER 0.
<9>:::=INCB WON'T WORK ALL THE TIME. FOR EXAMPLE, IF LOWER BYTE OF REGISTER 0 CONTAINS 377, THEN INCB WILL NOT DO THE SAME THING AS INC.
QUESTION # AND QUESTION
2. LOCATION LABEL CONTAINS A POINTER TO A LOCATION WHICH CONTAINS A NUMBER. REPLACE THAT NUMBER WITH ITS TWO'S COMPLEMENT.
HOW MANY TRIES? 4
GIVE A RIGHT ANSWER
NEG@LABEL<2>
GIVE A RIGHT ANSWER

GIVE A WRONG ANSWER
<NOTNEG><NOTLAB>
GIVE A WRONG ANSWER
<SUBER><NOTLAB><COMMA><ZERO><35>
GIVE A WRONG ANSWER
<SUBER><ZERO><COMMA><NOTLAB><35><4>
GIVE A WRONG ANSWER

RESPONSE AFTER 4 INCORRECT RESPONSES.
<32>
RESPONSE FOR STUDENT'S "HELP"
<20>
RESPONSE FOR STUDENT'S "HELP"
<25>
RESPONSE FOR STUDENT'S "HELP"

RESPONSE FOR STUDENT'S "GIVE UP"
<32>
<NOTNEG>:::=NEGB<13>↑COM<33>↑NEG↑COMB<33><13>
<NOTLAB>:::=@LABEL↑(LABEL)<34>↑LABEL<27>
<ZERO>:::=#0↑0<36>
<35>:::=THAT WILL FIND THE TWO'S COMPLEMENT BUT PUT IT NOWHERE THAT YOU COULD FIND IT!!! TRY PUTTING THE RESULT BACK WHERE IT BELONGS.
<32>:::=HOW ABOUT

NEG @LABEL ?

<25>:::=IF THE CORRECT OPERAND WAS IN LOCATION LABEL, THEN THE ANSWER WOULD BE TO APPLY A CERTAIN FUNCTION TO THE ADDRESS LABEL, I.E.

OPERATION LABEL

BUT THIS PROBLEM TELLS US THAT THE ADDRESS OF THE CORRECT OPERAND IS IN LOCATION LABEL, AND THEREFORE, THE CORRECT ANSWER IS OF THE FORM:

OPERATION @LABEL

NOTE THAT THERE DOES NOT EXIST AN ADDRESSING MODE (LABEL).
<13>:::=WHY BY BYTES? THAT'S SILLY!
<33>:::=COMPLEMENT COMPLEMENTS ALL THE BITS OF THE OPERAND. THIS FINDS THE ONE'S COMPLEMENT, NOT THE TWO'S COMPLEMENT.
<34>:::=THAT IS NOT A VALID ADDRESSING MODE. DEFERRED RELATIVE MODE IS ALWAYS OF THE FORM @LABEL, AND NEVER (LABEL).
<27>:::=<25>
<36>:::=IF THIS INSTRUCTION WORKED, YOU'D STILL ALSO NEED A NUMBER SIGN ON THE CONSTANT 0.
QUESTION # AND QUESTION

QUIT! ARE YOU SURE? YES
.RU COMPIL
GIZMO COMPILER VER IV-0 6/72
ID: T*0201
LESSON: EXMPL2
.

APPENDIX B. SAMPLE DIALOGUE BETWEEN STUDENT AND STUDENT MODE

```
GIZMO STUDENT MODE VER IV-0 6/72
PLEASE IDENTIFY YOURSELF: JOE STUDENT
LESSON NAME: EXMPL2
1. WRITE A PDP-11 ASSEMBLY LANGUAGE INSTRUCTION THAT ADDS
THE NUMBER ONE TO THE CONTENTS OF REGISTER 0.
ADDB #1,REGO          ;I HOPE THIS ADDS 1 TO REGISTER 0.
THE OPERATOR IS NOT A VALID ONE.  BETTER CHECK YOUR HANDBOOK.
THAT IS NOT A VALID WAY TO EXPRESS REGISTER 0.
THAT'S NOT VERY EFFICIENT.
TRY AGAIN.
ADD 1 RO
ALL CONSTANTS(LITERALS) MUST BE PREFIXED BY A # SIGN.
THE OPERANDS IN A 2-ADDRESS INSTRUCTION ARE SEPERATED BY A
COMMA.  YOU LEFT IT OUT.
THAT'S NOT VERY EFFICIENT.
TRY AGAIN.
HELP
FIND AN APPROPRIATE 1-ADDRESS INSTRUCTION FROM PAGE 92
IN THE PDP-11 HANDBOOK.
TRY AGAIN.
INCREMENT RO
TRY AGAIN
ADD 1,RO
ALL CONSTANTS(LITERALS) MUST BE PREFIXED BY A # SIGN.
THAT'S NOT VERY EFFICIENT.
THE BEST ANSWER IS

        INC RO              ;ADD ONE TO REGISTER ZERO

BUT

        ADD #1,RO

ALSO WORKS, BUT IS LESS EFFICIENT.

2. LOCATION LABEL CONTAINS A POINTER TO A LOCATION WHICH CONTAINS
A NUMBER.  REPLACE THAT NUMBER WITH ITS TWO'S COMPLEMENT.
COM (LABEL)
COMPLEMENT COMPLEMENTS ALL THE BITS OF THE OPERAND.  THIS
FINDS THE ONE'S COMPLEMENT, NOT THE TWO'S COMPLEMENT.
THAT IS NOT A VALID ADDRESSING MODE.  DEFERRED RELATIVE
MODE IS ALWAYS OF THE FORM @LABEL, AND NEVER (LABEL).
TRY AGAIN.
NEG @LABEL
VERY GOOD.

END OF LESSON EXMPL2.
.
```

APPENDIX C. INTERNAL STRUCTURE OF SOURCE LESSON

```
S I;GHE/Q 1. WRITE A PDP-11 ASSEMBLY LANGUAGE INSTRUCTION THAT ADDS
THE NUMBER ONE TO THE CONTENTS OF REGISTER 0.
←#4
←+ADD#1,<REG0><1><3>
←+SUB#177777,<REG0><1><12><3>
←+INC <REG0><2>
←+<WORD><10><12>
←-<ADDER><ONER><COMMA><NOTREG>
←-<SUBER><NONER><COMMA><NOTREG><1>
←-<INCER><NOTREG>
←-<SUBER><NOTREG><COMMA><NONER><1><4>
←-<ADDER><NOTREG><COMMA><ONER><1><4>
←*<19>
←H<20>
←G<19>
←<REG0>::=%0↑R0
←<WORD>::=.WORD 5200↑5200<11>
←<ADDER>::=ADDB<5>↑ADD
←<ONER>::=#1↑1<6>
←<COMMA>::=,↑<7>
←<NOTREG>::=REG0<8>↑REG<8>↑R0↑0<8>
←<SUBER>::=SUBB<5>↑SUB
←<NONER>::=#177777↑177777<6>
←<INCER>::=INCB<9>↑INC
←Q 2. LOCATION LABEL CONTAINS A POINTER TO A LOCATION WHICH CONTAINS
A NUMBER. REPLACE THAT NUMBER WITH ITS TWO'S COMPLEMENT.
← #4
← +NEG@LABEL<2>
← -<NOTNEG><NOTLAB>
← -<SUBER><NOTLAB><COMMA><ZERO><35>
← -<SUBER><ZERO><COMMA><NOTLAB><35><4>
← *<32>
← H<20>
← H<25>
← G<32>
← <NOTNEG>::=NEGB<13>↑COM<33>↑NEG↑COMB<33><13>
← <NOTLAB>::=@LABEL↑(LABEL)<34>↑LABEL<27>
← <ZERO>::=#0↑0<36>
← QUIT
```

Comments:

1. In the actual file, lesson parts are separated by control characters. I have indicated their presence here with a left pointing arrow (←).

2. Note that the single character following the ← is sufficient information for the compiler or editor to know what lesson part is presently being looked at. That is, Q for question, # for number of tries, + for correct answer, - for incorrect answer, * for message after X tries, G for GIVEUP, H for HELP, and < for a definition.

3. Since the standard teletype keyboard does not include the vertical bar, we use the up pointing arrow (↑) to indicate "or".

4. A letter "U" following the Q for question indicates end of lesson.

APPENDIX D. INTERNAL STRUCTURE OF OBJECT LESSON

```
S I;GHE/Q 1. WRITE A PDP-11 ASSEMBLY LANGUAGE INSTRUCTION THAT ADDS
THE NUMBER ONE TO THE CONTENTS OF REGISTER 0.
←#4
←+ADD#1,<%0↑R0>//1///3/
←+SUB#177777,<%0↑R0>//1///12///3/
←+INC <%0↑R0>//2/
←+<.WORD 5200↑5200//11/>//10///12/
←-<ADDB//5/↑ADD><#1↑1//6/><,↑//7/><REG0//8/↑REG//8/↑R0↑0//8/>
←-<SUBB//5/↑SUB><#177777↑177777//6/><,↑//7/><REG0//8/↑REG//8/↑R0↑0//8/>/
/1/
←-<INCB//9/↑INC><REG0//8/↑REG//8/↑R0↑0//8/>
←-<SUBB//5/↑SUB><REG0//8/↑REG//8/↑R0↑0//8/><,↑//7/><#177777↑177777//6/>/
/1///4/
←-<ADDB//5/↑ADD><REG0//8/↑REG//8/↑R0↑0//8/><,↑//7/><#1↑1//6/>//1///4/
←*//19/
←H//20/
←G//19/
←Q 2. LOCATION LABEL CONTAINS A POINTER TO A LOCATION WHICH CONTAINS
A NUMBER. REPLACE THAT NUMBER WITH ITS TWO'S COMPLEMENT.
←#4
←+NEG@LABEL//2/
←-<NEGB//13/↑COMB//33///13/↑COM//33/↑NEG><(LABEL)//34/↑@LABEL↑LABEL//27/
>
←-<SUBB//5/↑SUB><(LABEL)//34/↑@LABEL↑LABEL//27/><,↑//7/><#0↑0//36/>//35/

←-<SUBB//5/↑SUB><#0↑0//36/><,↑//7/><(LABEL)//34/↑@LABEL↑LABEL//27/>//35/
//4/
←*//32/
←H//20/
←H//25/
←G//32/
←QUIT
```

Comments:

1. Note that all occurrences of <NUMBER> have been replaced with //NUMBER/ to indicate to student mode that it should record the number as a message to be printed to the student later.

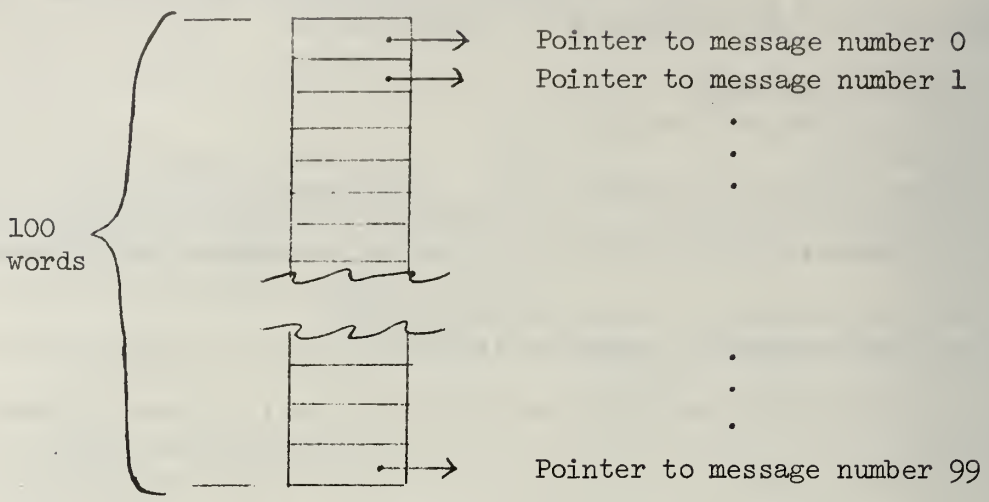
2. Note that all bracketed symbols have been expanded into their definitions.

3. Note that the actual definitions have been deleted.

4. Note that a special sequence of control characters have been inserted at the lesson beginning to tell student mode that this is in fact a GIZMO object lesson.

APPENDIX E. INTERNAL STRUCTURE OF COMMENT MODULE

First record in file:



The above table permits fast access to particular messages appearing in later records. Contained in each entry of the table is a unique pointer to each message consisting of the record (or block) number and the offset in bytes from the beginning of that record.

Other records in file:

```
-THAT'S NOT VERY EFFICIENT.
-INC R0 IS BETTER. BUT YOURS IS O.K. TOO.
-YOU'RE TRYING TO BE A WISE GUY.
-VERY GOOD.
-O.K. BUT A PSEUDO-OP IS NOT REALLY A PROPER PAL INSTRUCTION.
-THIS IS NOT THE 360/. THE DESTINATION ADDRESS APPEARS
ON THE RIGHT, NOT ON THE LEFT.
-THE BEST ANSWER IS

          INC R0          ;ADD ONE TO REGISTER ZERO

BUT

          ADD #1,R0

ALSO WORKS, BUT IS LESS EFFICIENT.
-FIND AN APPROPRIATE 1-ADDRESS INSTRUCTION FROM PAGE 92
IN THE PDP-11 HANDBOOK.
-IT'S A DEFAULT PSEUDO-OP.
-THAT OPERATOR IS NOT A VALID ONE.  BETTER CHECK YOUR HANDBOOK.
-ALL CONSTANTS(LITERALS) MUST BE PREFIXED BY A # SIGN.
```

-THE OPERANDS IN A 2-ADDRESS INSTRUCTION ARE SEPERATED BY A COMMA. YOU LEFT IT OUT.
-THAT IS NOT A VALID WAY TO EXPRESS REGISTER 0.
-INCB WON'T WORK ALL THE TIME. FOR EXAMPLE, IF LOWER BYTE OF REGISTER 0 CONTAINS 377, THEN INCB WILL NOT DO THE SAME THING AS INC.
-THAT WILL FIND THE TWO'S COMPLEMENT BUT WILL PUT IT NOWHERE THAT YOU COULD FIND IT!!! TRY PUTTING THE RESULT BACK WHERE IT BELONGS.
-HOW ABOUT

```
        NEG @LABEL      ?
```

-IF THE CORRECT OPERAND WAS IN LOCATION LABEL, THEN THE ANSWER WOULD BE TO APPLY A CERTAIN FUNCTION TO THE ADDRESS LABEL, I.E.

```
            OPERATION LABEL
```

BUT THIS PROBLEM TELLS US THAT THE ADDRESS OF THE CORRECT OPERAND IS IN LOCATION LABEL, AND THEREFORE, THE CORRECT ANSWER IS OF THE FORM:

```
            OPERATION @LABEL
```

NOTE THAT THERE DOES NOT EXIST AN ADDRESSING MODE (LABEL).
-WHY BY BYTES? THAT'S SILLY!
-COMPLEMENT COMPLEMENTS ALL THE BITS OF THE OPERAND. THIS FINDS THE ONE'S COMPLEMENT, NOT THE TWO'S COMPLEMENT.
-THAT IS NOT A VALID ADDRESSING MODE. DEFFERRED RELATIVE MODE IS ALWAYS OF THEFORM @LABEL, AND NEVER (LABEL).
-//25/
-IF THIS INSTRUCTION WORKED, YOU'D STILL ALSO NEED A NUMBER SIGN ON THE CONSTANT 0.

APPENDIX F. TEACHER'S MANUAL

Writing a lesson for GIZMO is easy...and fun! Here are the simple step by step instructions. In all examples shown, text which is underlined has been generated by GIZMO through the teletype; all text which is not underlined has been entered by the user or lesson author. The symbol ↵ indicates the depression of the "carriage return" or "return" key on the teletype.

I. Definitions of terms:

A. BNF - The Backus-Naur form algebraic language described by J. W. Backus at the 1959 ACM-GAMM conference in Zurich.

B. BNF-like language - This is the modification to the original BNF language which is used by GIZMO lesson authors. It is explained fully in this manual.

C. Bracketed symbol - This is one component of the GIZMO BNF-like language. In terms of formal grammars, it is equivalent to a non-terminal symbol. In terms of the BNF meta-language, it is "the class of" the symbol. In plain english, it is equivalent to a word which either has already been defined or else will be defined later but in either case, it must be defined. There are two types of bracketed symbols in GIZMO. There are "answer matching" bracketed symbols which are always one to six characters long, starting with a letter (A-Z), and surrounded by brackets. Examples: <A>, <ANSW1>, <A1113>, <CINCH>, <Z>, <MONKEY>, <CAT5>, etc. The second type is called a "message number" bracketed symbol. It is always either one or two digits (0-9) surrounded by brackets. Examples: <43>, <2>, <0>, <99>, <63>, etc. Their uses will be explained later.

D. Comment (or COM) file - One of two lesson files created by teacher mode. It need not be compiled. After the source lesson file is compiled into the

object lesson file, the comment and object lesson files together comprise a lesson which is student usable. The comment file contains a list of all the messages that can be outputted to the student.

E. Compile - The process of changing the source lesson, which is easily readable and editable, into an object lesson, which is easily interpretable by the student mode.

F. ETS - Educational Timesharing System is the operating system under which GIZMO presently operates.

G. Lesson compiler - This is the second of three programs which comprise the GIZMO CAI system. It's job is to translate the source lesson into the object lesson. See compile.

H. Lesson title - This is the 1 to 6 character title assigned to a lesson by the lesson author which he and his students use whenever accessing any of the three lesson modules (SRC, COM, or OBJ).

I. Object (or OBJ) lesson file - This is the easily interpretable lesson module which is outputted from the lesson compiler and is used by student mode to administer a lesson to a student.

J. Option - Any one of many traits that an author of a lesson may choose to apply to an entire lesson--never to a part only. They are all characterized by a single letter followed by 0-3 characters dependent on the particular option. See a later section for a full explanation.

K. Source (or SRC) lesson file - This is the easily read and edited lesson module which is outputted from the teacher mode and edited by an editor. It cannot be used to administer a lesson to a student until it has been compiled into an object lesson file by the lesson compiler.

L. Student mode - This is the third of three programs which comprise the GIZMO CAI system. It's job is to accept an object lesson file and a comment file and administer a lesson to a student.

M. Teacher code word - This is the 6 character identification word assigned to each teacher and required by the teacher mode on lesson compiler before access to a lesson is permitted. It maintains security by preventing unauthorized personnel from destroying or creating lessons.

N. Teacher mode - This is the first of three programs which comprise the GIZMO CAI system. It is used by the lesson author to create a lesson.

O. Text - This is any information typed into the lesson by the lesson author during lesson creation that appears within a specification of an answer or definition of a bracketed symbol but is not itself a bracketed symbol or a part of the BNF-like language.

II. Initial dialogue with ETS.

This will give control from the ETS monitor to the GIZMO teacher mode.

```
.HE 2 7 DUMBO7↵
.SI 5↵
.RUN TEACH↵
```

III. Initial dialogue with GIZMO teacher mode.

This will identify you as a valid lesson author and your lesson name as a valid lesson name.

```
GIZMO TEACHER MODE VER.IV 6/72
ID: XXXXXX↵
LESSON: YYYYYY↵
```

where XXXXXX is your teacher code word and YYYYYY is your chosen lesson name.

IV. The BNF-like language for GIZMO.

A. Syntax

The syntax used in this language is similar to that of the BNF metalanguage, with some additions and modifications:

1. <A> means "the class of A" or in other words "the class of strings which are defined as A." The name may be from 1 to 6 characters starting with a letter.
2. ::= means "is defined as"
3. | means "or"
4. * means "an arbitrary number of occurrences of the preceeding class"
5. <n> means give message #n to the student $0 \leq n \leq 99$

Thus, for example,

<DIGITS>::=0|1|2|3|4|5|6|7|8|9

means that the class of DIGITS is defined to be any one of the 10 characters 0, 1, 2, 3, 4, 5, 6, 7, 8, or 9.

<INTEGR>::=<DIGITS>*

means that the class of INTEGR is defined to be an arbitrary number of DIGITS. Similarly,

<ODDINT>::=<INTEGR><ODDIG>|<ODDIG>
<ODDIG>::=1|3|5|7|9

means that ODDINT is the class of odd integers, namely any integer followed by an odd digit, or just an odd digit.

B. Semantics

The BNF-like language is used in two places. When specifying a right or wrong answer to teacher mode, it is assumed that you are defining the class of right or wrong student answers respectively and therefore it

is necessary to type only the right side of the definition sentence. When defining bracketed symbols, teacher mode tells you which symbol you are defining and therefore once again it is necessary to type only the right side. If when specifying a student answer or a definition of a bracketed symbol you write,

<A> <1> <B> <2> | <C> <3> <D> <4>

it means a student may write any answer which complies with the definition of the bracketed symbol A followed by the definition of the bracketed symbol B, or complies with the definition of the bracketed symbol C followed by the definition of bracketed symbol D. If the student's answer complies with one of these two, then he will be given messages 1 and 2 or 3 and 4 respectively. If however, the student's answer starts with a string from the definition of A but is not followed by a string from the definition of B, he will be given only message 1. The same holds true for C and D and message 3.

V. Creating a lesson

GIZMO will ask you a series of questions after each of which he will wait for your responses. GIZMO will check your answers for correct syntax and will give you the question

WHAT?

if your response is wrong. Whenever GIZMO says WHAT?, you should retype the last line correctly as it was not acceptable in its present state.

A. The first question will be:

OPTION:

This asks you to indicate which of the available options you would like to have implemented while the student is taking your lesson. The available options are:

1. Evaluator. This option specifies that all numeric expressions in the student's response will be evaluated as PAL expressions and then their values will be compared to the numbers indicated in your answers. For example, if you indicate MOV #5, R0 as a correct answer and evaluator is set, then MOV #5-10.+3+7, R2-2 would be graded as correct. To specify option evaluator, type:

E↵

2. Ignorer. This option specifies that only up to a certain character in the student's response should be examined and that all characters typed afterwards will be ignored. For example, if you indicate that the correct answer is MOV #3, R0 and you have also indicated ignorer-semicolon, then

MOV #3, R0 ; COMMENT

would be graded as correct. To specify option ignorer, type:

IX↵

where X is the character as defined above.

3. Scruncher. This option specifies that all occurrences of a certain character in the student's response should be skipped. For example, if you indicate the correct answer is MOV #3, R0 and you have also indicated scruncher-blank, then

MOV #3 , R0

would be graded as correct. To specify option scruncher, type:

SX↵

where X is the character to be scrunched. However, if you specify a character in <u>your</u> answer that is also scrunchable, that character will be considered required in the student's answer to produce a correct match.

4. Brackets. This option specifies that the teacher wishes to redefine the bracket types used in the BNF-like language. This should be used only if the teacher intends to use the default-type brackets (namely < >) for another purpose within the lesson. To specify option brackets, type

BXX↵

where XX are the two new characters of the teacher's choice to be used as brackets.

5. Repetition. This option specifies that the teacher wishes to redefine the symbol in the BNF-like language that means "arbitrary number of occurrences of." The default symbol is *. This should be used only if the teacher wants to use the *, when immediately following a bracketed symbol to mean the character *. Note that the teacher may use * anywhere in the lesson that he wishes to mean the actual character * (except immediately following a bracketed symbol) without specifying this option. To specify option repetition, type:

RX↵

where X is the character that the teacher wishes to use following all repetitive labels. For example, if

R&↵

was specified, then to define an integer, the teacher would later type:

<INTEGR>::=<DIGIT>&
<DIGIT>::=0↑1↑2↑3↑4↑5↑6↑7↑8↑9

6. Give-Up. This option indicates that the teacher wants to give a response to the student if he types "GIVE-UP." If specified here,

teacher mode will request a message for this after each question entered. To specify give-up type:

G↵

7. Help. This option is the same as give-up except that the messages will be given the student when he types "HELP." If specified here, teacher mode will request a message for this after each question entered. The teacher may allow the student to ask for help as many times as helpful. To specify help, type:

H↵

8.* Timer. This option permits the teacher to specify a time-limit for the student to complete his response. To specify timer, type:

TXX↵

where XX is the two-digit number in decimal seconds ($00 \leq XX \leq 99$) requested.

9. Quick. This option is used to request "quick lesson writing." That is, instructions are omitted at each step of teacher mode and only a one-letter code is used. Do not use this unless experienced at lesson writing!

The following codes are used:

O for option

Q for Question # and Question

for How many tries?

R for Right answer

W for Wrong answer

* for Response after X incorrect responses.

G for Response to student's GIVE-UP

H for Response to student's HELP

To specify option quick, type:

Q↵

Note: For all the options above, you must give the correct number of parameters (0 for E, 1 for I, 1 for S, etc.). You may indicate as many options as you wish. After each option entry, GIZMO will ask you the same question again. To terminate option entries, type a null entry, namely

↵

If you specify the same character to be scrunched and ignored, the character will only be scrunched.

B. The next question will be:

QUESTION # AND QUESTION

This asks you to indicate the text you would like to have printed to the student. If you want to continue on the next line, terminate the previous line with a line feed instead of the usual carriage return. If you wish to terminate the lesson simply type a null entry.

C. The next question will be:

HOW MANY TRIES?

This asks you to indicate how many times you want the student to be allowed to guess wrong answers before giving him the correct answer or any other message that you wish. You may type as many characters here that you wish but only the first will be examined. Therefore, it must be an integer less than ten, and greater than zero. If not, GIZMO will type WHAT? and you should try again.

*Not available on GIZMO Version IV. It will be available on later versions.

D. The next question will be:

GIVE A RIGHT ANSWER

This asks you to indicate an answer which should be graded as correct by GIZMO student mode. It must be syntactically and semantically correct as defined in section IV of this manual. If not, GIZMO will type WHAT? and you should compare your response with the correct ones as defined above and then try again.

GIZMO will continue to ask for right answers until you type a null entry.

E. The next question will be:

GIVE A WRONG ANSWER

This asks you to indicate an answer which should be graded as incorrect by GIZMO. The same rules apply here as for right answers.

F. The next question will be:

RESPONSE AFTER X INCORRECT RESPONSES

This asks you to indicate what message you wish to have printed to the student if he fails to answer the question correctly after the number of times that you have indicated above. This can be a series of message numbers correctly bracketed and/or an actual textual message. Thus, you may write

<2>BUT YOU MUST REALIZE HOWEVER THAT <3>.

which will mean message 2, followed by the above words followed by message 3, followed by a period.

G.**. The next question will be:

MESSAGE AFTER STUDENTS "HELP"

This asks you to indicate a message you wish to have printed to the student if he types HELP. The same rules apply here as for section F above. This question will be asked repeatedly to enable the teacher to give a student different help on each successive plea to receive it. To go on with next question, type a null entry.

H.** The next question will be

MESSAGE AFTER STUDENTS "GIVE-UP."

This asks you to indicate a message you wish to have printed to the student if he types GIVE-UP. The same rules apply here as with F and G above. However, it will only be asked once, since a student obviously can't give-up repeatedly.

I. GIZMO will now ask you to define those symbols that you have used in this question but were not previously defined by you. Note, by the way, that all definitions given in this section will be global to the entire lesson. All symbols must be defined in terms of "the BNF-like language" as described above.

VI. Terminating the lesson

When you type ↵ in response to a QUESTION # AND QUESTION, you will be asked to verify this action. In response to

QUIT! ARE SURE?

type YES↵.

The COM file is now ready for student's use but the SRC file must be compiled by the lesson compiler to translate it into the OBJ file which then, in conjunction with COM, defines an entire lesson which is student usable. To compile the lesson:

**These will be asked only if you have previously specified options G or H.

```
. RU COMPIL
GIZMO LESSON COMPILER
ID:  XXXXXX
LESSON:  YYYYYY)
```

BIBLIOGRAPHIC DATA SHEET	1. Report No. UIUCDCS-R-73-563	2.	3. Recipient's Accession No.

4. Title and Subtitle

The Design, Implementation and Analysis of a Computer-Assisted Instruction System on a Mini-Computer

5. Report Date: January, 1973

6.

7. Author(s): Alan Mark Davis

8. Performing Organization Rept. No.

9. Performing Organization Name and Address

Computer Science Department
University of Illinois
Urbana, Illinois 61801

10. Project/Task/Work Unit No.

11. Contract/Grant No.: US NSF GJ812

12. Sponsoring Organization Name and Address

National Science Foundation
Washington, D.C.

13. Type of Report & Period Covered

14.

15. Supplementary Notes

16. Abstracts

GIZMO is a mini-computer CAI system designed for use in an introductory assembly language course taught on a PDP-11. It provides an easy method for teachers to create lessons, as well as for students to learn assembly language by taking the lessons. It is written in PAL-11 assembly language.

17. Key Words and Document Analysis. 17a. Descriptors

CAI
Computer Assisted Instruction
Mini-Computers

17b. Identifiers/Open-Ended Terms

17c. COSATI Field/Group

18. Availability Statement	19. Security Class (This Report)	21. No. of Pages
Release unlimited	UNCLASSIFIED	73
	20. Security Class (This Page) UNCLASSIFIED	22. Price